A Prototype Tool Set for Interworkings

Sjouke Mauw, Thijs Winter

The interworking language provides a graphical method for the description of synchronous interactions between system components. In this paper we give an overview of the language and discuss the prototype tool set which has been developed to support the use of interworkings.

1. Introduction

Telecommunications software is huge, real-time and distributed. Adding new features to a system can be very difficult because of the possible interactions with existing features. As the number of features grows, and correspondingly the number of feature interactions, specialised description techniques are required to manage the complexity problem. The technique presented in this paper is based on interworkings.

Interworkings are used in the analysis phase of the development process at PKI (Philips Kommunikations Industrie) Nürnberg for describing the message interactions between blocks. The specification of one functional block contains the most common sequences of message interactions with all the neighbour blocks it communicates with. Interworkings are a synchronous variant of message sequence charts [4]. Message sequence charts have been used for a long time by CCITT Study Groups in their recommendations and within industry, according to different conventions and under various names. They are used, for example, for the specification of radio communication systems. Interworkings also play an important role in other areas than the telecommunication industry. Several object-oriented methods have absorbed Message Sequence Charts (ObjectOry [5], Rumbaugh [9]). In close cooperation with PKI Nürnberg a project is being performed in which an Interworking Tool Set (ITS) is specified and prototyped that can process, analyse, and combine interworkings.

Compared to other trace languages interworkings have the advantage of a clear graphical layout and structuring. However, interworkings are only suitable for the description of relatively small parts of the system behaviour. Therefore, in order to describe the behaviour of a system more completely, composition operations on interworkings similar to the composition operations in for example LOTOS are required.

A collection of interworkings describes the behaviour of a system on a high level of abstraction. Each interworking is a projection of a part of the communication behaviour of a system onto a set of entities. In a telecommunication context where for example SDL is used as description language, an entity may be a process or a set of processes combined into one functional block. Interworkings are event oriented instead of entity oriented and only synchronous communication actions can be modeled. This means that messages can not be delayed as in message sequence charts where communication is asynchronous.

The full interworking language is described in [7]. In [6] we explore techniques from

process algebra [2] to define the formal semantics.

In this paper we first give an introduction to interworking diagrams and operators. After that we discuss the Interworking Tool Set. We will not explain the whole interworking language, nor the complete functionality of the tools.

2. Interworkings

2.1. Interworking diagrams

The basic graphical language of interworking diagrams is very simple and resembles the syntax for basic sequence charts [8]. Vertical lines are used to represent entities within a system and horizontal arrows are used to represent communications between two entities. No global time axis is assumed for one interworking. Along each vertical entity-line the

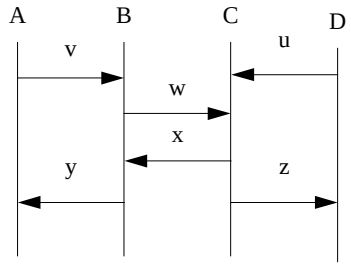

Figure 1. Example interworking diagram

time runs from top to bottom, however, no proper time scale is assumed. Events of different entities are ordered only via messages. Within interworkings, no delay between a message output and the corresponding message input is explicitly modelled. Therefore, an interworking imposes only a partial ordering on the events being contained. In Fig. 1, the messages `u` and `v` are not related and no ordering exists between them. This also holds for the messages `y` and `z`.

A collection of interworkings describes the behaviour of a system on a high level of abstraction. Each interworking describes a common sequence of communication actions performed by the contained entities. Note that the interworking does not specify communication actions with entities that are not contained. The behaviour of a single entity is thus only partially described by the events being contained in the interworking. The actual behaviour will be an interleaving with communication events concerning other entities. Between the output of `v` and the input of `y`, entity `A` in Fig. 1 for example can have interactions with an entity `E` which is not in the diagram. However, it can not have a communication action with either one of the entities `B`, `C`, or `D` between the output of `v` and the input of `y`.

2.2. Interworking sequencing

An interworking diagram can be constructed from atomic communication actions and applications of the interworking sequencing operator (denoted by $\circ_{iw}$). The interworking sequencing, or simply the sequencing of two interworking diagrams is the vertical

concatenation of the two diagrams (See Fig. 2). In this case, where the interworking dia-

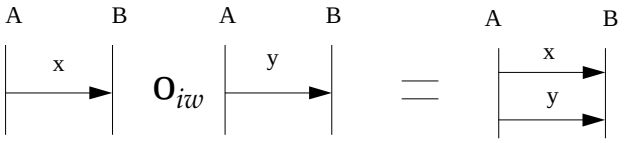

Figure 2. Sequencing

grams have all entities in common, the sequencing corresponds to a real sequentialisation in time. The operands need not necessarily have all entities in common. In the next example (Fig. 3) the two interworkings only have entity `B` in common. If interworkings

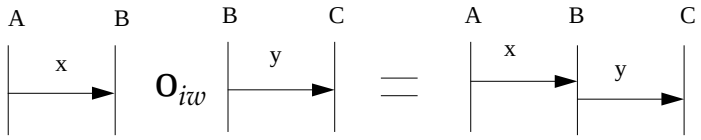

Figure 3. Sequencing interworkings with one entity in common

contain disjoint entities, the behaviour described by means of their sequencing will in most cases not be equal to simple sequential composition of the behaviours. Events that do not involve the same entity or are not related via messages will be unordered in the sequential composition. An example of the sequencing of two interworkings that contain unrelated messages is given in Fig. 4. Note that the right-hand side describes a single interworking in which the vertical position of the arrows does not indicate any ordering.

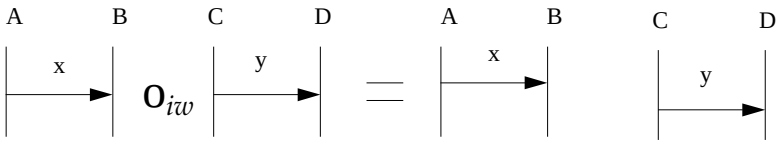

Figure 4. Sequencing interworkings with disjoint entities

2.3. Interworking merge

The interworking merge, or simply called the merge (denoted by $\|_{iw}$), of two interworkings is their interleaved composition with the restriction that the interworkings are forced to synchronise on a set of communication actions. This set consists of the communication actions concerning every pair of entities which the interworkings have in common. In

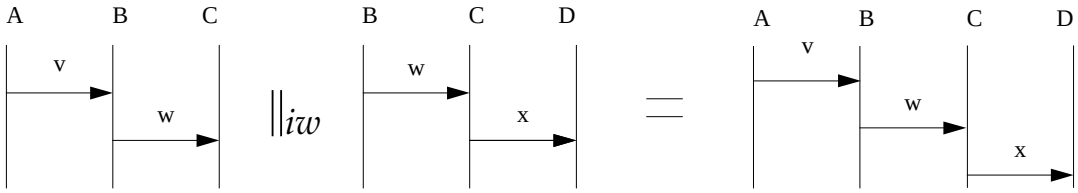

Figure 5. Merge

Fig. 5 two interworkings are merged, which have entities `B` and `C` in common. In those cases where the interworking diagrams describe disjoint entities, the merge is equal to real parallel composition. An example of the merge of two interworking diagrams in which disjoint entities are described is given in Fig. 6. Note also that the merge of these two interworking diagrams is equal to the sequencing of these diagrams (See Fig. 4).

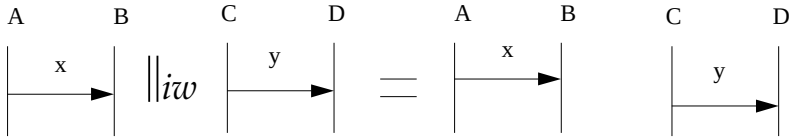

Figure 6. Merging interworkings with disjoint entities

2.4. Consistency

If two interworkings are merged, the interworkings have to synchronise on all communication actions between entities which they have in common. In case this synchronisation succeeds, we will call the two interworkings *merge-consistent*. However, if two interworkings have multiple entities in common but do not synchronise on all the communication actions between these entities then the interworkings are not merge-consistent and the merge of the interworkings will contain so called deadlock or inaction. The two interworking diagrams in Fig. 7 for example are not merge-consistent.

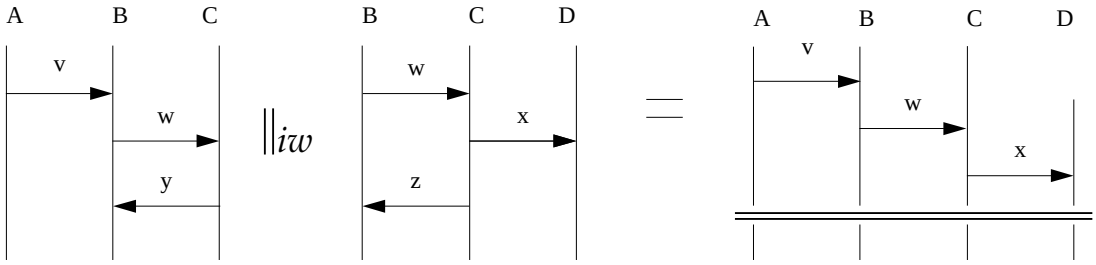

Figure 7. Inconsistent interworkings

2.5. Refinement

When using stepwise refinement in developing a system, one might want to consider the system at several levels of abstraction. Therefor we define the notion of refinement for interworkings. The intuition is that the behaviour of a single entity within an interworking can be refined into a collective behaviour of a number of entities. In Fig. 8 entity `B` is refined into two entities `B1` and `B2`, which interact with actions `x`, `y` and `z`.

Figure 8. Example of refinement

3. The prototype toolset

In order to check the consistency between different functional specifications, tools have to be developed that can process, analyse and combine the interworkings. A typical user specifies several components of a system with the use of the textual format for interworkings. Such a specification may contain applications of the sequencing and merge operator, or special features such as macro's. Using the tools one can for example check the syntax, expand macro's, generate diagrams and check if the interworkings are consistent.

The Interworking Tool Set is a prototype which has been built using the Elegant compiler generator [1]. The prototype comprises of three parts, the interworking processor (IWP), the intermediate language compiler (ILC) and a term-rewriting system (TRS). The architecture of the prototype is given in Fig. 9.

Figure 9. Architecture of prototype

Together these parts provide the following functionality.

- Static analysis
- Macro expansion
- Interworking diagram generation (PostScript and ASCII)
- Test sequence generator
- Parallel combination of interworkings
- Sequential combination of interworkings
- Refinement analysis.

The interworking processor performs static analysis and macro expansion on interworkings in textual format. It also translates interworkings in textual format into PostScript or ASCII diagrams or into test sequences in the GST (Genau und Sinnvoll Testen) format [3]. Additionally, the tool IWP translates basic interworkings into an intermediate language T that serves as input for the two other tools. More details are given in Fig. 10.

Figure 10. Interworking Processor

The intermediate language compiler translates interworkings and compositions thereof to the intermediate format. The tool ILC can also check if a certain interworking is a refinement of another interworking. More details are given in Fig. 11.

Figure 11. Intermediate Language Compiler

The term-rewriting system normalises interworkings in an intermediate format, using the definitions of the interworking sequencing operator and interworking merge operator. The TRS is the heart of the prototype toolset and most of the essential algorithmics are performed according to the term rewriting paradigm. More details are given in Fig. 12.

Figure 12. Term Rewrite System

4. Conclusions

Interworking diagrams have been used already for several years, turning out useful for systems specification and for test purposes. These diagrams were not standardized and did not have a proper semantics. For larger applications it shows tedious to manually check for inconsistencies. Also it is hard to change a specification, because updating the drawings manually is expensive. The formal basis for interworkings and the Interworking Tool Set based upon it, shows useful as a design method, which helps overcoming these problems.

Currently we have a prototype of the Tool Set, which is being evaluated. There are options for improvements of the output layout, and of the user-interface. Also, in practice it could be useful to consider slightly modified definitions, for example to allow merging of interworkings not covering the same time period. Of course it is important to follow the external developments relating interworkings with other features from the field of MSCs. These are asynchronous communication, message overtaking and conditions. These and other options are subject of discussions and investigations in which IST-SDR (sector Infor-

mation and Software Technology of Pilips Research Eindhoven, department Specification, Design and Realisation) and PKI-PRC (Philips Kommunikations Industrie, department Public Radio Communications) participate. The process of transforming the prototype toolset into a true tool has started in the project "PRC Methodology Upgrade", which is part of the IST-SDR contract research programme.

The formal algebraic semantics, together with the use of the Elegant compiler generator provided a straightforward prototyping strategy for tool builders. This easy prototyping has helped in evaluating the use of interworkings in the aforementioned Interworking project.

REFERENCES

1. L. Augusteijn, The Elegant Compler Generator Tool Set, Release 5, Philips Research Laboratories Eindhoven, 1991.
2. J.C.M. Baeten & W.P. Weijland, Process algebra, Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, ISBN 0 521 40043 0, 1990.
3. H. Botzek, GST-A Benutzerhandbuch, Version 1.0, Philips Koomunikations Industrie AG, Kommunikationssysteme, 30-Jul-1992.
4. CCITT, Recommendation Z.120, Message Sequence Charts (MSC). 1991.
5. I. Jacobson, a.o., Object-Oriented Software Engineering, A Use Case Driven Approach, Addison Wesley, ISBN 0 201 54435 0, 1992.
6. S. Mauw, M. van Wijk & T. Winter, A formal semantics of synchronous interworkings, Sixth SDL Forum, Darmstadt, 1993.
7. S. Mauw, M. van Wijk & T. Winter, Syntax and semantics of synchronous interworkings, Philips IST technical report RWB-508-RE-92436, 1992.
8. E. Rudolph, Syntax and Semantics of Basic Sequence Charts, Contribution to the Study Group X, WP X/3, Q8, CCITT meeting, Geneva 1992.
9. J. Rumbaugh, a.o., Object-Oriented Modeling and Design, Prentice Hall International, ISBN 13 630054 5, 1991.